TRALE Definite Clauses

Grammar Engineering, SS 2006

Georgiana Dinu

Overview

- Definite clauses: Overview

Overview

- Definite clauses: Overview
- Clauses in Prolog Language

Overview

- Definite clauses: Overview
- Clauses in Prolog Language
- TRALE Definite Clauses
  - Defining a clause
  - Relational Attachments

Definite Clauses

- (1) $\phi_1 \wedge \ldots \wedge \phi_n \longrightarrow \phi_0$

  $\phi_0 \quad :- \phi_1, \ldots, \phi_n$

  $head :- body$

  (2) ϕ_0

- TRALE syntax

```
<clause> ::= <literal> if <goal>.
<literal> ::=  <pred_sym>
            |  <pred_sym>(<seq(desc)>)
```

- **member2** as a definite clause in TRALE:

```
member(X, hd:X) if true.
member(X, tl:Xs) if member(X, Xs).
```

Definite Clauses in Prolog: Declarative Semantics

- ```
  concatenate([], L, L).
  concatenate([X|L1], L2, [X|L3]) :- concatenate(L1, L2, L3).

  ?- concatenate([a], [b], [a,b]).

  concatenate([a], [b], [a,b]) :- concatenate([], [b], [b]).
  ```

- *A goal is true if it is the head of some clause instance and each of the goals (if any) in the body of that clause instance is true, where an instance of a clause (or term) is obtained by substituting, for each of zero or more of its variables, a new term for all occurrences of the variable.*
  ```

Definite Clauses in Prolog: Declarative Semantics

- ```
  concatenate([], L, L).
  concatenate([X|L1], L2, [X|L3]) :- concatenate(L1, L2, L3).

  ?- concatenate([a], [b], [a,b]).

  concatenate([a], [b], [a,b]) :- concatenate([], [b], [b]).
  ```

- *A goal is true if it is the head of some clause instance and each of the goals (if any) in the body of that clause instance is true, where an instance of a clause (or term) is obtained by substituting, for each of zero or more of its variables, a new term for all occurrences of the variable.*
  ```

Definite Clauses in Prolog: Procedural Semantics

- Query

  ```
  ?- concatenate(X, Y, [a,b]).
  ```

- Instantiated queries

  ```
   ?- concatenate(X1, Y, [b]).
   ?- concatenate(X2, Y, []).
  ```

- Solutions

  ```
  X = [a,b] Y = []

  X = [a]   Y = [b]

  X = []    Y = [a,b]
  ```

- If there is no matching head for a goal, the execution backtracks to the most recent successful match.

The Cut Symbol

- The cut operation commits the system to all choices made since the parent goal was invoked, and causes other alternatives to be discarded.
- ```
  member(X, [X|_]).
  member(X, [_|L]) :- member(X, L).

  | ?- member(X, [d,e,f]).
  ```
  Solution: d, e, f.
- ```
  member(X, [X|_]) :- !.
  member(X, [_|L]) :- member(X, L).

  | ?- member(X, [d,e,f]).
  ```
  Solution: d
- ```
  ?- member(e, [d, e, f]) .

  Yes
  ```
  ```

If then else

- `P -> Q ; R`

  Analogous to if P then Q else R and defined as if by

  ```
  (P -> Q; R) :- P, !, Q.
  (P -> Q; R) :- R.
  ```

- Only explores the first solution to the goal P (removes all choice-points created by P and executes Q.)

- `P -> Q`

  When occurring as a goal, this construction is read as equivalent to:

  ```
  (P -> Q; fail)
  ```

- Example

  ```
  add(X,L1,L2) :- member(X,L1) -> L2 = L1 ; L2 = [X|L1].
   1 ?- add(a, [b,c], [a,b,c]) .
   Yes
   2 ?- add(X, [b,c], [a,b,c]) .
   No
  ```

Negation by failure

- Fails if the goal P has a solution, and succeeds otherwise. This is not real negation ("P is false"), but a kind of pseudo-negation meaning "P is not provable". It is defined as:

```
\+(P) :- P, !, fail.
\+(_).
```

- The only time we get something like the desired result if there is no existentially quantified variable in the goal.
- Example

```
p(a).
p(b).
q(c).

?-q(X), \+p(X).    % succeeds with X=c
?-\+p(X), q(X).    % fails
```

TRALE Definite Clauses

- Terms: TRALE Descriptions
- Evaluation of a definite clause query
  - Solution to Queries: Satisfiers of the entire query as most general solutions
  - Example

    ```
    | ?- query member(X, [noun, verb]).
    ```

  - Solution 1:

    ```
    member([0] noun
           CASE case,
           ne_list
           HD [0]
           TL ne_list
              HD verb
                 VFORM vform
              TL e_list)
    ```

TRALE Definite Clauses

- Terms: TRALE Descriptions
- Evaluation of a definite clause query
  - Solution to Queries: Satisfiers of the entire query as most general solutions
  - Example

    ```
    | ?- query member(X, [noun, verb]).
    ```

  - Solution 2:

    ```
    member([0] verb
           VFORM vform,
           ne_list
           HD noun
              CASE case
           TL ne_list
              HD [0]
              TL e_list)
    ```

TRALE Definite Clauses: Syntax

<goal> ::= true
| <literal>
| prolog(<prolog_goal>)
| (<goal>,<goal>)
| (<goal>;<goal>)
| (<desc> =@ <desc>)
| (<cut_free_goal> -> <goal>)
| (<cut_free_goal> -> <goal> ; <goal>)
| !
| (\+ <goal>)
| when(cond, <goal>)

TRALE Definite Clauses: Syntax

```
<goal> ::= true
        | <literal>
        | prolog(<prolog_goal>)
        | (<goal>,<goal>)
        | (<goal>;<goal>)
        | (<desc> =@ <desc>)
        | (<cut_free_goal> -> <goal>)
        | (<cut_free_goal> -> <goal> ; <goal>)
        | !
        | (\+ <goal>)
        | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | (\+ <goal>)
         | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | (\+ <goal>)
         | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | (\+ <goal>)
         | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | (\+ <goal>)
         | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | (\+ <goal>)
         | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
        | <literal>
        | prolog(<prolog_goal>)
        | (<goal>,<goal>)
        | (<goal>;<goal>)
        | (<desc> =@ <desc>)
        | (<cut_free_goal> -> <goal>)
        | (<cut_free_goal> -> <goal> ; <goal>)
        | !
        | (\+ <goal>)
        | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

<goal> ::= true
| <literal>
| prolog(<prolog_goal>)
| (<goal>,<goal>)
| (<goal>;<goal>)
| (<desc> =@ <desc>)
| (<cut_free_goal> -> <goal>)
| (<cut_free_goal> -> <goal> ; <goal>)
| !
| (\+ <goal>)
| when(cond, <goal>)

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | ( \+ <goal>)
         | when(cond, <goal>)
```

TRALE Definite Clauses: Syntax

```
<goal> ::= true
         | <literal>
         | prolog(<prolog_goal>)
         | (<goal>,<goal>)
         | (<goal>;<goal>)
         | (<desc> =@ <desc>)
         | (<cut_free_goal> -> <goal>)
         | (<cut_free_goal> -> <goal> ; <goal>)
         | !
         | (\+ <goal>)
         | when(cond, <goal>)
```

Co-routining 1

- Syntax

```
| when(<cond>,<goal>)
```

- when/2 delays execution until some event is witnessed.
- Example:

```
append(X,Y,Z) if
   when(  ( X=(e_list;ne_list)
          ; Y=e_list
          ; Z=(e_list;ne_list)
                ),
            undelayed_append(X,Y,Z)).

undelayed_append(L,[],L) if true.
undelayed_append([],(L,ne_list),L) if true.
undelayed_append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).
```

Co-routining 2

- Conditional descriptions syntax

```
<cond> ::= <variable>^(<cond>)
         | <quantified_cond>

<quantified_cond> ::= <quantified_cond>,<quantified_cond>
                    | <quantified_cond>;<quantified_cond>
                    | <variable>=<cond_desc>

<cond_desc> ::= <variable>
              | <type>
              | max(<type>)
              | <feat>:<cond_desc>
              | <path> == <path>
              | <cond_desc>, <cond_desc>
              | <cond_desc> ;<cond_desc>
```

Co-routining 3

- Shared variables in conditionals

```
when(X=([f]==[g]),bar(X))


when(X=(f:Y,g:Y),bar(X))
```

- Narrow scope

```
when(Y^(X=(f:Y,g:Y)),bar(X))

foo(X) if
 Z = f:Y,
 when(Y^(X=(f:Y,g:Y)),bar(Y,Z))
```

Attaching Definite Relations: Constraints

- Syntax

```
CondDesc *> Desc goal Goal.
```

- If a goal is specified in a constraint, the constraint is satisfied only if the goal succeeds.
- Variables occuring in the consequent are bound with scope that extends over the consequent and over the relation attachments.
- Example

```
phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:HeadSubcat)
goal
   append(PhrSubcat,[Synsem],HeadSubcat).
```

Attaching Definite Relations: Phrase Structure Rules 1

- Syntax

```
<rule_clause> ::= cat>  <desc>
                 |cats> <desc>
                 |goal  <goal>
```

- Example

```
backward_application rule #
 (synsem:..., qstore:Qs)
 ===>
 cat> (synsem:..., qstore:Qs1),
 cat> (synsem:..., qstore:Qs2),
 goal>  append(Qs1,Qs2,Qs).
```

Attaching Definite Relations: Phrase Structure Rules 2

Evaluation

- evaluated in the order they are specified
- all possible solutions are found and the resulting instantiations are carried over to the rule

Example

```
schema2 rule #
(cat:(head:Head,subcat:[SubjSyn]))
goal> three_or_less(Comps),
cats> Comps,
cat>  (cat:(head:Head,
       subcat:[Subj|Comps])).

three_or_less([]) if true.
three_or_less([_]) if true.
three_or_less([_,_]) if true.
three_or_less([_,_,_]) if true.
```

Attaching Definite Relations: Lexical Rules

- Syntax:

```
<lex_rewrite> ::= <desc> **> <desc> if <goal>
```

- ```
  passive_lex_rule ##
  (word,
   synsem:...,
   arg_st:[(loc:(cat:...,
                 cont:Cont2)), Synsem1|List])
   **>
  (word,
   synsem:...,
   arg_st:([Synsem1|List];Result))
  if append(([Synsem1|List]),[(loc:(cat:...,
                                    cont:Cont2))],Result)
                                    morphs
  X becomes X.
  ```
  ```

Summary

- TRALE's definite clause extension provides a way of encoding HPSG relations

Summary

- TRALE’s definite clause extension provides a way of encoding HPSG relations
- Various computational aspects become important in:
  - Formulating definite clauses
  - Attaching them as relational constraints

Summary

- TRALE's definite clause extension provides a way of encoding HPSG relations
- Various computational aspects become important in:
  - Formulating definite clauses
  - Attaching them as relational constraints
- Further reading:
  - On Prolog definite clauses: Sicstus Prolog Manual
    - Prolog Language: *http://www.sics.se/sicstus/docs/latest/html/sicstus.html/Prolog-Intro.html#Prolog%20Intro*
    - Predicate Index: *http://www.sics.se/sicstus/docs/latest/html/sicstus.html/Predicate-Index.html#Predicate%20Index*

Summary

- On defining TRALE clauses: TRALE User Manual Chapter 5
  http://www.ale.cs.toronto.edu/docs/man/ale_trale_man/index.html
  Co-routining: 5.2
  Shared variables in conditionals: 5.2.1
- On attaching goals: TRALE User Manual Chapters 4,6
  Constraints: T4.3
  Phrase structure rules: 6.4.1
  Lexical Rules: 6.3
- On evaluating definite clause queries: Trale User Manual Appendix B, Section 2.4
- On source level debugger: Trale User Manual Appendix B, Section 2.10